Program Generators

They're not as easy to use as some advertising copy suggests.

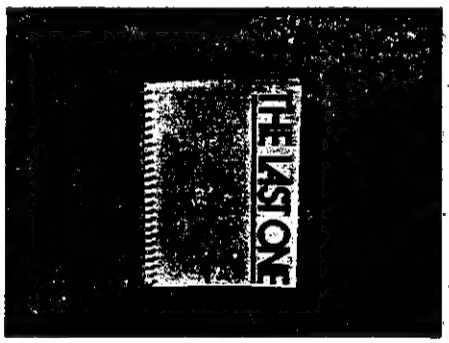
George Stewart
Technical Editor

Would you like to be able to tell your computer what you want it to do without ever having to learn a programming language? Well, you can. You simply tell your computer what you want in layman's terms, and it figures out how to accomplish your wish and creates a program to do it.

The software tools that perform this feat are called program generators or application generators. (Technically, a program generator creates a *stand-alone* program that you can list, store, copy, and use. An application generator, on the other hand, generates a software package that is dependent on the application generator: to run your generated program, you use a *run-time* portion of the application generator. In this article, I'll use the term *program generator* to include both kinds.)

The first program generators were written for mainframe computers back in the late 1960s. Their purpose was to increase the productivity of data-processing departments. Now several have been announced for microcomputers and, for the first time, are being aimed at nonprogrammers as well as professional programmers.

Photos by Katherine Coker



The most heavily and boldly advertised package is The Last One, $600 from D. J. 'AI' Systems Ltd. One typical ad starts out with the headline "Your prayers have been answered." Understandably, the promotion has produced considerable skepticism and controversy in the computer community. Another product introduced with less fanfare is Quic-N-Easi, $395 from Standard Microsystems Inc. I'll use these two products as examples in this background report on program generators. (For more specifics on each product, see the text boxes.)

How Program Generators Work

Program generators are *problem-oriented* rather than *procedure-oriented*. In other words, because the program generators possess information about common programming problems like keyboard entry, file input/output, and data sorting, they let you concentrate on the problem you're trying to solve rather than on the special computer procedures required to solve it.

Let's say you want a program that creates a mailing-list file on a floppy disk. First, of course, you must decide exactly how you want information stored in that list—even a manual, paper-based system requires that much. Do you want to store the names in alphabetical order or by member ID? Last name first or vice versa? What's the longest name and how many lines are in the address?

You must also specify the exact steps for inputting and storing names. You would have to do much the same thing if you were explaining your wishes to another person instead of to a computer. Table 1 summarizes the

At a Glance

Name
The Last One

Type
Program generator

Manufacturer
D. J. 'Al' Systems Ltd.
Two Century Plaza, Suite 480
2049 Century Park East
Los Angeles, CA 90067
(213) 203-0851

Price
$600

Format
8-inch or 5¼-inch floppy disks

Language
BASIC

Computer
CP/M systems, Radio Shack TRS-80 Model II, Apple II, Commodore PET

Documentation
Tutorial manual plus machine notes for specific computers

Audience
Nonprogrammers who want custom software

At a Glance

Name
Quic-N-Easi

Type
Program generator

Manufacturer
Standard Microsystems
136 Granite Hill Court
Langhorne, PA 19047
(215) 968-0689

Price
$395

Format
8-inch or 5¼-inch floppy disk

Language
Machine code

Computer
CP/M systems, Radio Shack TRS-80 Model III

Documentation
Three-ring binder containing tutorial, quick-reference card, and programmer's reference

Audience
Nonprogrammers who want custom software and programmers seeking to speed programming efforts

steps you might take in planning your mailing list.

Now how do you communicate all this information to a program generator? If the program generator is to be useful for nonprogrammers, most of your work should already have been done (in the planning phase just described). You'll probably communicate with the program generator in three phases: data description, screen design, and program procedure.

Data description tells the computer how many pieces of information (fields) exist in each logical file entry (record) and what kind of data goes into each. *Screen design* is the arrangement of headings and prompting messages that the operator will see on the screen. *Program procedure* tells the computer what to do with the data that is typed in. Data description and screen design are relatively straightforward, but the program design phase is where the program generator really shows its stuff (or lack of it).

After you've completed the program specification, the generator wi take care of the programming detail asking you for additional informatio whenever necessary. Figures through 4 and listings 1 and 2 sho uses of The Last One and Quic-N-Ea to specify the mailing-list applicatio

Using an ordinary programmin language, your task is far more i volved. The data description, scree design, and program procedure a must be coded in computer-langua statements covering a multitude details: how to create and initialize disk file, input each data item fro the keyboard, write each complete record to disk, etc. Including steps t handle errors (keyboard mistakes disk problems) is an especially i tricate and burdensome task. Instea of focusing on your problem in la man's terms, you must convert it int technical terms.

Evaluating Program Generators

In your evaluation, you shou look for capabilities in six gener areas: data entry, program logic spe

ification, file storage, report generation, calculations, and editing convenience.

Data entry: Getting information into the computer, as illustrated in the mailing-list application previously described, is the bread and butter of program generators. It's the simplest part of most programs and yet often the most tedious to program. A good program generator should allow easy creation of display forms, the screen layouts that prompt the operator for data. Photo 1 shows a typical display form. Ideally, you should be able to construct the display form on the screen, not on paper, and modify the screen-input form without modifying the entire application program. Checks for invalid entries should be provided automatically by the program generator.

Program logic specification: How hard is it to tell the computer what you want? That depends on how much knowledge is embedded in the program generator. To take a few minor examples, does the program generator know what alphanumeric data looks like (A–Z, a–z, 0–9, ., +, −), or do you have to make up a procedure to check the validity of each entry?

As a general rule, if an operation is generically repetitive (such as searching through a table for a specific entry), you shouldn't have to take great pains specifying the procedure to accomplish it. If you do, then you, not the program generator, are doing most of the programming work.

File storage: This is an important characteristic of program generators and it may take some careful study. Does the program generator allow both major types of file storage—sequential and random access? Sequential access allows you to read information in the same order in which

Description of each member record:
Last name (15 letters)
First name (10 letters)
Member ID (5 digits)
Date of last contact (8 characters as mm/dd/yy)
Street (25 characters)
City (20 letters)
State (2 letters)
Zip (5 digits)

General description of program operation:
1. Ask operator to type in a record (e.g., information for one member).
2. Check all entries for validity.
3. Write the information to the disk file.
4. Ask operator if there are member records to be entered; if there are, then repeat step 1.
5. If not, end the program.

Table 1: *Details of mailing list to be worked out before using the program generator. Numbers in parentheses are maximums for each item.*

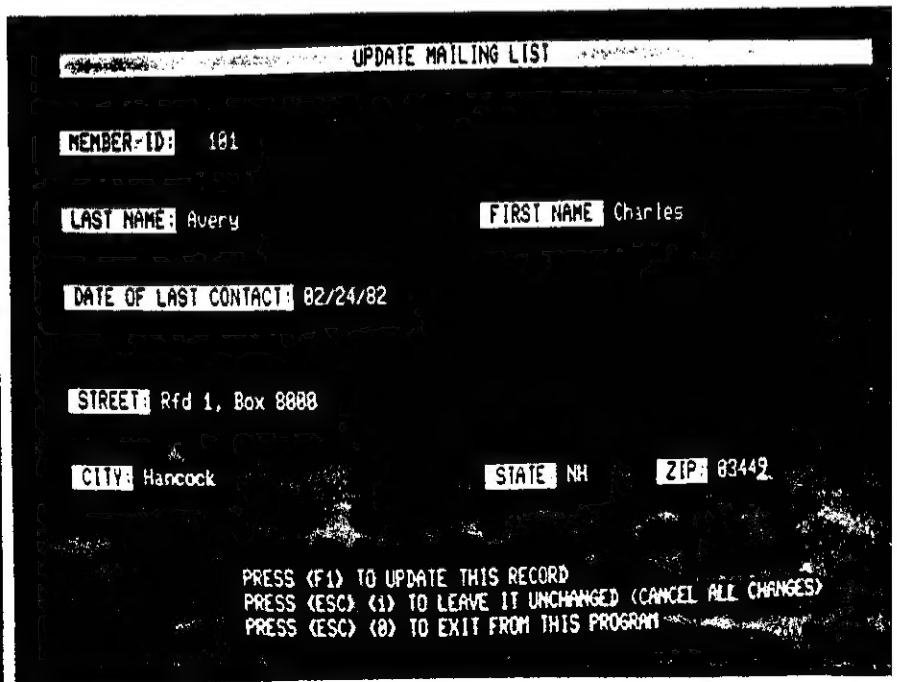


Photo 1: *A data-entry screen created with Quic-N-Easi.*

Field	Field Label	A/N/D	Field Size
1	LAST NAME	Alpha	15
2	FIRST NAME	Alpha	10
3	MEMBER ID	Numeric (0)	6
4	DATE OF LAST CONTACT	Date	8
5	STREET	Alpha	25
6	CITY	Alpha	20
7	STATE	Alpha	2
8	ZIP	Numeric (0)	6

Figure 1: *The data description for the mailing-list program using The Last One.*

```
1       .. Open MEMLIST file
2       .. Set pointer to the end of MEMLIST file
3       .. Keyboard input using labels from MEMLIST
4       .. Write data to MEMLIST file
5       .. Ask user < ANY MORE RECORDS TO ENTER >. If yes branch to 3
6       .. Terminate program
```

Figure 2: *The procedure description for the mailing-list program in the form of a general flowchart using The Last One.*

Listing 1: *A small part of the BASIC mailing-list program generated by The Last One. The total dialogue required to specify the program is not shown. The application specified here is for entering new records only; another program would be needed to update existing records.*

```
8  CLEAR 5000:DEFDBL N:DEFINT R:ONERRORGOTO60000
9 DIM AA$(8)
10 CLOSE:OPEN"D",1,"MEMLIST:1",92
11 FIELD 1,15 AS AA$(1),10 AS AA$(2),6 AS AA$(3),8 AS AA$(4),25 AS AA$(5),20 AS
AA$(6),2 AS AA$(7),6 AS AA$(8)
12 RC(1)=LOF(1)+1
13 CLS:X8=0:X9=0
14 PRINT@(1,26),"MAILING LIST CREATION PROGRAM"
15 PRINT@(6,1),"LAST NAME":PRINT@(6,12),STRING$(15,46):PRINT@(6,12),:LINEINPUTS
$:IFLEN(S$)<=15THEN17
16 PRINT@(6,12),SPC(LEN(S$)):GOTO15
17 LSET AA$(1)=S$
18 PRINT@(6,40),"FIRST NAME":PRINT@(6,53),".....":PRINT@(6,53),:LINEINPUTS
$:IFLEN(S$)<=10THEN20
19 PRINT@(6,53),SPC(LEN(S$)):GOTO18
20 LSET AA$(2)=S$
```

it was written. Random access allows reading, writing, or updating information in any order. For applications that require frequent updates of information scattered throughout a file, random access is almost a necessity. Some program generators offer a third kind of file access called indexed sequential. In effect, your data is sorted automatically as it is entered. Instead of referring to data in terms of arbitrary record numbers, you can refer to it in terms of filing keys. The same thing can be accomplished through random-access files, but *you* have to provide the indexing.

Another important feature is interactivity. Do all data files have to be explicitly named during program generation, or can the end user specify files at run time? For example, suppose you have generated a sorting program. Can the operator enter the name of the file to be sorted, or does the generated program have to know about it in advance? The answer to these questions will tell you much about the flexibility of a program generator.

Equally important, what *file structures* are available? Can information be defined in a hierarchy? In a mailing list, can a list of family members be grouped under "member name" or a list of previous addresses be grouped under "address"? You can write powerful applications programs more simply if the program generator has built-in facilities for such *hierarchical data*.

Report specification: When it comes to outputting results, is it easy to explain your desired report format to the program generator? You should not have to go to great length to have headings and subheadings inserted at the appropriate positions. If many columns are to be printed, you

shouldn't have to worry about squeezing them all in. The program generator should take care of that by breaking the column headings into two or more lines, etc.

Calculations: Most applications will require some calculations on the data: comparisons between names, arithmetic operations, etc. Obviously, you will need to understand the required operations fully before you can explain them to the program generator. However, you should be able to enter the necessary operations and formulas without resorting to computerese. For example, if you want to update an account balance, you should be able to accomplish this in a straightforward manner such as:

NEW BALANCE = OLD BALANCE − PAYMENT RECEIVED

You should *not* have to resort to for mulas like this:

V1 = V2 − V3

Editing convenience: This may b the most important aspect of a pro gram generator, since 60 to 80 percen of programming time is usually de voted to maintenance (modification of existing programs. Obviously

```
MEMBER ID: #####

LAST NAME: ###############                FIRST NAME: ##########

DATE OF LAST CONTACT: ##/##/##


STREET: ##########################

CITY: ####################                STATE: ##        ZIP: #####.

          PRESS <F1> TO UPDATE THIS RECORD
          PRESS <ESC> <1> TO LEAVE IT UNCHANGED (CANCEL ALL CHANGES)
          PRESS <ESC> <0> TO EXIT FROM THIS PROGRAM
```

Figure 3: *The screen layout for the mailing-list program using Quic-N-Easi. The "#" signs show the size and position of keyboard input fields.*

OFFSET	FIELD	LEN	R	C	DESC	JUST	FILL	MY-EN	MU-EN	MU-FL	MU-TB	PROC
0	ID	5	3	12	D	R		Y	Y	N	N	GETRECORI
1	LASTNAME	15	6	12	×	L		Y	N	N	Y	
2	FIRSTNAME	10	6	53	×	L		Y	N	N	Y	
3	MONTH	2	9	23	D	R		Y	N	Y	Y	
4	DAY	2	9	26	D	R		Y	N	Y	Y	
5	YEAR	2	9	29	D	L		Y	N	Y	Y	CHECKDATI
6	STREET	25	13	9	×	L		Y	N	N	Y	
7	CITY	20	16	7	×	L		Y	N	N	Y	
8	STATE	2	16	48	A	L		Y	N	Y	Y	
9	ZIP	5	16	63	D	L		Y	N	Y	Y	

Figure 4: *The data descriptions given to Quic-N-Easi. Column abbreviations used are LEN=field length, R=display rou C=display column, DESC=data description, JUST=justification (left or right), FILL=character, MY−EN=may ente MU−EN=must enter, MU−FL=must fill, MU−TB=must tab, and PROC=procedure associated with this field. As soon as th operator types in a member ID number, the GETRECORD procedure gets the member record, if it has been written. Th CHECKDATE procedure ensures that the operator enters a valid date as mm/dd/yy.*

Listing 2: *The procedures you must specify using the Quic-N-Easi language for the mailing-list application. In effect, you write this "program"; however, it is much shorter and simpler than an equivalent program written in an ordinary programming language. This application uses indexed sequential files and allows you to update existing records.*

```
 10: * FUNCTION KEY PROCEDURE
 10:          PROC KEY0
 10:          CLOSE 1
 10:          SYSTEM
 10:          END
 10:          PROC KEY1
 10:          UNLOCK ID
 10:          HOMECLEAR
 10:          END

100:          PROC GETRECORD
100:          IF ID GE 1 AND ID LE 200 THEN GOTO 10010
100:          ERROR "MEMBER ID NUMBER IS OUT OF RANGE: [1 - 200]"
100:          CLEAR ID
100:          RESUME
100:          END
100: 10010    POSN 1 TO ID
100:          READ 1: 10020
100:          GET * FROM 1
100:          LOCK ID
100:          END
100: 10020    UNLOCK ID
100:          END

200:          PROC LOAD
200:          OPEN "MEMLIST",3,97,1: 20010
200:          END
200: 20010    MAKE "MEMLIST",3,97,10,5,1: 20020
200:          END
200: 20020    ERROR "CAN'T CREATE FILE"
200:          SYSTEM
200:          END

300:          PROC ENTER
300:          UNLOCK ID
300:          RESTART 1
300:          PUT * TO 1
300:          WRITE 1: 30010
300:          SECURE 1: 30020
300:          END
300: 30010    ERROR "ERROR IN WRITING RECORD"
300:          CLOSE 1
300:          SYSTEM
300:          END
300: 30020    ERROR "ERROR IN SECURING FILE"
300:          CLOSE 1
300:          SYSTEM
300:          END

400:          PROC CHECKDATE
400:          IF MONTH LT 1 OR MONTH GT 12 THEN GOTO 40010
400:          IF DAY LT 1 OR DAY GT 31 THEN 40010
400:          END
400: 40010    ERROR "ERROR IN DATE FORMAT -- USE MONTH/DAY/YEAR"
400:          CLEAR MONTH
400:          CLEAR DAY
400:          CLEAR YEAR
400:          NEXT MONTH
400:          END
```

The Last One

The rather grandiose idea implicit in the product's name is that it will be the last program you ever need to buy. If that is indeed the case, the reason will probably be that you give up computing out of sheer frustration with The Last One.

The Last One is written entirely in BASIC. It consists of dozens of separate program and data files comprising more than 175,000 bytes of code (on the Model II version). In terms of sheer program size, D. J. 'AI' Systems is certainly giving you your money's worth. Of course, programs can't be rated solely in terms of cost per byte of code. Far more important is how useful the code is.

The Last One is a fully menu-driven system; that means at every stage of its operation, the screen lists currently available options. For example, when you start the program, you see the main dispersal menu:

Create program.....<1>
Modify program.....<2>
Modify file.....<3>
External files.....<4>
Enquiry.....<5>
Certify new disk.....<6>
Return to BASIC.....<7>

Each time you select an option, The Last One must load the appropriate program and run it. This makes the system quite sluggish. Most of your time is spent watching the computer display the message "Please wait...working." Because of a complex hierarchy of menus, skipping from one activity to another (from, for example, screen design to procedure specification) is tortuous if not impossible.

Sample Use

For generating routine data-entry applications (such as the mailing list described in the main article), The Last One is acceptable but cumbersome. You start by specifying exactly what information goes into the file. You assign a name to each field, describe the field (any characters, numeric only, or date-format data), and specify the field size. Having only three data types puts a larger burden on you to check data entries for valid information; often you will need to ensure that the data falls into a much narrower category (compare with Quic-N-Easi). If you make a mistake, you can correct it by retyping all the information for the affected field.

After describing the data, you set a "file pointer," which determines the position in the file where input/output will begin. This is probably the first place where previous computer knowledge is useful.

Next you specify the program logic in two steps. Using a "flowchart creation menu," you select the desired sequence of operations for your program. Figure 2 shows the steps used to program the mailing-list application. When you're done with the flowchart, you have a very general description of the program logic. However, most of the work is yet to be done.

The next phase is called "coding," and it's by far the most tedious. All the generalities of the flowchart must be turned into specific procedures. Wherever you have indicated a branch (change in program flow), you now specify the destination of the branch, referring back to the original flowchart. Wherever you have specified "input from keyboard" in the flowchart, you will now be prompted to design an input screen. The Last One doesn't have a full-featured screen editor, so you must locate the prompting fields using row and column numbers. To change a completed screen, you must erase it and start all over.

Wherever you have indicated calculations, The Last One will ask you to specify them as formulas. Unfortunately, you cannot use the field names but must resort to meaningless symbols like V1, V2, V3, etc.

Outputting results is similar to keyboard input: you specify the output format by relating data fields to various rows and columns on the screen. The Last One will go through the entire list of variables in your program and ask where each one of them is to be output. Typically, only a very few of them are desired as output. This means much needless effort.

When the generalities have all been reduced to specifics, The Last One will generate a BASIC program. The final result will contain routines to handle keyboard and disk-related errors. You will be able to use the program (and associated data files) independently of The Last One.

Should you ever want to modify the program, you'll probably want to use The Last One again, even if you know BASIC. The reason is that the generated program is completely undocumented. Variable names used have no meaning, and no explanatory remarks are embedded in the program. You can have a copy of the flowchart included at the beginning of the program, but that is too general to be really helpful in program modification. It's easy to modify a flowchart, and generating the flowchart isn't difficult in itself. The hard part is modifying the coded program. Rather than changing a few parts and leaving the rest of the coding unchanged, you must painstakingly repeat the entire coding procedure.

Documentation

The instruction manuals are tutorial and quite readable. One describes the package in general; the other describes specifics related to the machine you are using. The general manual takes you step by step through a simple mailing-list application, the best way to get you into the subject. However, the manual is not organized for easy reference. Information is scattered about in different sections, and much information is too abbreviated.

Summary

The Last One does contain a considerable amount of embedded knowledge. It can generate a great deal of BASIC code given a few simple commands. Unlike other program generators, The Last One doesn't require you to learn a specification language. It's a shame that the system isn't faster and easier to use.

If you are willing to wade through a tedious maze of menus and specification procedures that may take hours, and if you refuse to learn BASIC or any other programming language, you can probably find a use for The Last One, especially if your application is to perform simple data storage and retrieval. . . . **G. S.**

Quic-N-Easi

The instruction manual describes Quic-N-Easi as "an applications development language that dramatically reduces development time and produces more professional, clearer screen presentations." Compared to The Last One's hyperbole, this is a refreshingly modest and accurate description.

To use this product successfully, you will need to learn some programming concepts and the Quic-N-Easi language. That's going to take a while (anywhere from a day to a week or more). But once you've learned it, you have a tool that really can speed up the programming of common business applications. (If you learn the advanced features, you can go far beyond run-of-the-mill data entry and retrieval applications.)

Quic-N-Easi consists of two programs, the "format builder" and the "run-time interpreter." The format builder lets you describe the data, set up screen formats, and specify procedures in the Quic-N-Easi language. The result of this effort is called a format file and is really your application program. The run-time program interprets the format file and, in so doing, performs the application.

The striking features of Quic-N-Easi are ease of editing and logical operation. You work with only three "modes" or activities while creating a format file: building a format background (designing the screen layout), defining the data fields, and specifying program procedures. It's simple to skip from one activity to another without losing work in either area.

Sample Use

To generate the sample mailing-list application (see main article), you start by defining the screen format. Quic-N-Easi has a screen editor that makes this easy: rather than referring to screen locations with row and column numbers, you type the desired information right onto the screen. Next you locate operator-entry fields on the screen and specify what kind of data goes in each. There are nine categories of data, including various combinations of numbers, uppercase letters, and lowercase letters. This generous selection takes much of the burden of data checking from you during the procedure specification stage: the computer will automatically ensure that valid data is entered from the keyboard.

Finally, you specify procedures to be executed immediately after each field is entered or after the entire screen form is filled. Listing 2 shows the procedures used to generate the mailing-list application.

The Quic-N-Easi language is simpler than BASIC; nevertheless, it is a computer language. If it's your first, expect some difficulty. One good thing about this language is that you can always refer to your data in terms of names you choose—like ID, STREET, etc. This is true even when you're specifying calculations to be performed on these fields. Only for internal calculations do you have to resort to names like #S0, #N0, and #B0.

Disk input/output is another strong point of this product. In addition to sequential and random-access files, Quic-N-Easi offers indexed sequential files, which enable your file update programs to operate with exceptional speed. (See section entitled "File Storage" page 42.) The mailing-list application was programmed using indexed sequential files.

Documentation

The instruction manual for Quic-N-Easi contains a self-teaching guide and a programmer's reference section. The self-teaching guide uses prepared format files supplied with the software distribution disk and makes an effective introduction to the system. Mastering the system is going to take quite a while, and the programmer's reference section will become useful as you begin to grasp the principles of operation. A handy reference card is also provided.

Summary

Quic-N-Easi will not free you from the task of programming. To make full use of it, you will need to understand fundamental principles of programming—as well as learn the Quic-N-Easi language. However, once you've passed these hurdles, Quic-N-Easi should help you to generate common business-application programs much faster than could be done using BASIC or other programming languages. And the level of expertise required to create a given application with Quic-N-Easi is lower than that required to create the same application in an ordinary language. **G. S.**

programming concepts and techniques.

One final point seems obvious but is often missed: to use a program generator effectively, you must fully understand the desired application. It's no use, for example, trying to make a program generator produce a double-entry ledger system if you know nothing about accounting. Even the mailing-list application requires that you have a good understanding of the best way to store data. (How many characters should be allowed for the name field, address field, etc?) Manual systems are much more flexible than computer systems in these areas; you'll probably have to do more specific planning th you're used to.

One thing's for sure. Using an a plication generator will give y more appreciation for the work pr grammers do. If using a program ge erator takes so much effort, thi about what programmers have to through.■